View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Purdue E-Pubs

Purdue University
Purdue e-Pubs

Department of Computer Graphics Technology Degree Theses

Department of Computer Graphics Technology

7-24-2012

User Assisted Tree Reconstruction from Point Clouds

William P. Leavenworth II
Purdue University, wleavenw@purdue.edu

Follow this and additional works at: http://docs.lib.purdue.edu/cgttheses

Part of the Graphics and Human Computer Interfaces Commons

Leavenworth, William P. II, "User Assisted Tree Reconstruction from Point Clouds" (2012). *Department of Computer Graphics Technology Degree Theses.* Paper 11.
http://docs.lib.purdue.edu/cgttheses/11

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By William P. Leavenworth II

Entitled
User Assisted Tree Reconstruction from Point Clouds

For the degree of Master of Science

Is approved by the final examining committee:

Bedrich Benes
Chair

David Whittinghill

James Mohler

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Bedrich Benes

Approved by: Marvin Sarapin 7/17/2012
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

User Assisted Tree Reconstruction from Point Clouds

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

William P. Leavenworth II

Printed Name and Signature of Candidate

7/19/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

USER ASSISTED TREE RECONSTRUCTION FROM POINT CLOUDS

A Thesis

Submitted to the Faculty

of

Purdue University

by

William P Leavenworth II

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2012

Purdue University

West Lafayette, Indiana

For Mom, Dad, Melody, and Ashly.

ACKNOWLEDGEMENTS

Foremost, I would like to thank my graduate committee: Dr. Bedrich Benes, for his constructive feedback and frank perspectives on computer graphics research; Dr. James Mohler, for providing graduate school guidance when it seemed I was going off track; and Dr. David Whittinghill, for helping fund three semesters of my research. A big thank-you goes out to my undergraduate advisor, Dr. Tracy Camp, for encouraging me to pursue research in the first place.

I would also like to thank Juraj Vanek, for writing the implementation's engine; Sören Pirk, for providing the LIDAR tree scans; Baoquan Chen, for providing the synthetic tree models; Ziv Yaniv, for writing the RANSAC functions; and the contributors at Visual Computing Lab, ISTI, CNR, for distributing the MeshLab tool.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Appendix Figure

ABSTRACT

Leavenworth, William P. M.S., Purdue University, August 2012. User Assisted Tree Reconstruction from Point Clouds. Major Professor: Bedrich Benes.

LIDAR is a useful tool for quickly digitalizing real world objects, but it usually takes some effort to produce a recognizable object from the raw input. When the object is a tree, the challenge is to create a three-dimensional model that represents its general shape, while avoiding the influence of undersampling and noise. In the method developed for this research, the user creates a sketch overlaying a display of the raw input data. Each node in the sketch creates an estimated slice of the tree skeleton at that point, and the slices are connected according to the connectivity of the sketch. Both user-guided and automated sketches are tested. Analysis is performed by simulating the LIDAR scan process on pre-existing three-dimensional tree models, and then comparing the surface of the reconstruction to that of the original. The results of the method are presented on scans of both synthetic and real trees. It is shown that the output meshes follow the general shapes of the trees, although the influence of undersampling and noise can still be found.

CHAPTER 1. INTRODUCTION

This chapter introduces the problem addressed by this research, as well as its scope and significance. Major research questions are answered. The assumptions, limitations, and delimitations are outlined. Lastly, a list of key terms is defined.

1.1 Statement of Problem

This research addresses the reconstruction of individual tree models from unordered 3D point sets. These sets, or point clouds, are acquired by a discrete scan from outside the tree, such as with LIDAR. The potential of incorporating user sketches in the reconstruction process is explored. After the user sketches a set of connected lines onto an interface, generalized cylinders are produced to give a 3D model representation of the original tree.

1.2 Research Questions

This research focuses on a single primary and secondary research question.

1.2.1. Primary

Can user sketches aid the reconstruction of biological trees from unordered point sets?

1.2.2. Secondary

Would more complex sketches yield more accurate results?

1.3 Scope

The reconstruction methodology is limited to models of biological plants. Data points are obtained by using a virtual LIDAR simulator on a set of previously generated plant meshes, which have a wide variety of shapes and complexity. Because the focus is on accurately reconstructing the tree's general shape, no attempt has been made to synthesize leaves and twigs; these detract from accuracy because they are an attempt to patch up areas of undersampling. Lighting, shading, and animation are ignored for the sake of this study. Applications are limited to those that require the rendering of scenes in real-time, such as computer games and learning simulations. Finally, level of detail relative to view distance is not explicitly tested.

1.4 Significance

Trees are a significant feature in nearly every environment on earth. It is desirable to be able to simulate them in virtual environments, but due to their complex shapes and wide variety of appearances, this is not a trivial task. Much work has been done in using formal grammars, such as L systems, to procedurally generate natural-looking trees (Prusinkiewicz & Lindenmayer, 1990). However, some applications would benefit from the capture and reproduction of existing plants. In ecology, this would help automate the study of tree growth patterns in forests (Pfeifer, Gorte, & Winterhalder,

2004). In forestry, having an approximation for crown density and trunk size help assess forest fire risk (Morsdorf et al., 2004) or wood volume for timber (Cheng, Zhang, & Chen, 2007). Other applications include landscaping, gaming, and simulation.

1.5 Assumptions

This research was performed under the following assumptions:

1. The input files for the scanning procedure represent models of complete, individual trees.
2. In the input files, models are represented with the bottom of the trunk or stem at the lowest point.
3. The results given by the current hardware will generalize to any other high performance computer equipped with a modern GPU.

1.6 Limitations

This research was performed under the following limitations:

1. Only two computer configurations were used for testing.
2. Models were scanned from the outside, meaning occlusion errors may exist.
3. Results may not generalize to every species of plant.

1.7 Delimitations

This research was performed under the following delimitations:

1. This technique was designed for reconstructing features of a bare tree. Foliage is ignored.
2. Level of detail techniques were not implemented or measured.
3. Lighting and texturing were not included in the performance measurements.
4. For testing purposes, scanned data sets were taken from virtual models of individual trees, not from field data.
5. No user study was written or performed to evaluate the visual fidelity of the resulting images.

1.8 Definitions of Key Terms

Allometry – Rules concerning the thickness of tree limbs, based partly on strength and flow requirements. (Xu, Gossett, & Chen, 2007)

Alpha Shape – "A mathematically well-defined generalization of the convex hull. Its result is a series of subgraphs of the Delaunay triangulation, depending on different [user-defined] alpha values." (Zhu, Zhang, Huand, & Jaeger, 2008)

Generalized Cylinder – "A 3D object representation comprising a plane or a curve, called the 'cross-section', that moves along an axis, called the 'spine', according to a 'sweeping rule'. During the sweeping, the spine decides the orientation of the cross-section and the sweeping-rule decides the way the cross-section changes." (Pan & Lane, 1999)

Hausdorff Distance – "Measures the extent to which each point of a 'model' set lies near some point of an 'image' set and vice versa. Thus, this distance can be used to

determine the degree of resemblance between two objects that are superimposed on one another." (Huttenlocher, Klanderman, & Rucklidge, 1993)

L System – A formal grammar for modeling the shape and growth of organisms, particularly plants. (Prusinkiewicz & Lindenmayer, 1990)

Level of Detail – "The real-time 3D computer graphics technique in which a complex object is represented at different resolutions and the most appropriate representation chosen in real time in order to create a tradeoff between image fidelity and frame rate." (Luebke et al., 2003, p. 338)

LIDAR – Light detection and ranging. A scanning machine fires laser pulses at physical objects to sample their distances. The collection of samples can serve as a 3D digitalization of real-world scenes. (Côté, Widlowski, Fournier, & Verstraete, 2009)

RANSAC – Random sample consensus. An iterative algorithm for determining the parameters to a model from a data set, designed to be highly resilient to noise. (Fischler & Bolles, 1981)

Skeleton – Locus of centers of maximal spheres inside an object. (Cornea Silver, Yuan, & Balasubramanian, 2006)

1.9 Summary

This chapter introduced the problem of reconstructing trees from point clouds. It detailed the research questions, scope, and significance. Key terms were defined, and the assumptions, limitations, and delimitations were given.

CHAPTER 2. LITERATURE REVIEW

This chapter provides a summary of relevant literature. It begins with looking at research in extracting tree features from LIDAR scans. Work in reconstructing object skeletons is examined next, followed by methods that specifically reconstruct trees. Literature in building trees from user sketches is examined briefly. Existing methods for analyzing reconstructed models are covered. Finally, concluding remarks on the state of the field are given.

2.1 Extracting Tree Features from Scan Data

Pyysalo and Hyyppa (2002) use data from an airborne laser scanner to reconstruct tree crowns and extract key features. The data produces a map of points with varying heights. Points with the lowest z coordinates are classified as ground points and used to create a digital terrain model (DTM). The rest of the points create a digital surface model (DSM) to represent the trees. Individual trees are segmented from the DSM based on height valleys. Features extracted from each tree include tree height, crown height, and the average distance of points from the trunk at different heights. Holmgren and Persson (2003) extend this method to classify trees in a forest by species. After extracting features from an input map, it identifies those features that most

clearly distinguish the species. In future maps, tree species could be classified using this training data. 95% accuracy is the result for the classification for Scots pine and Norway spruce. There are some misclassification errors due to the natural variation in different members of a tree species. Also, while the method of Pyysalo and Hyyppa (2002) are found to give acceptable results, noticeable errors occur at significant tree overlap.

Morsdorf et al. (2004) present a method for reconstructing individual trees from LIDAR scans of forest scenes. A raw point set, scanned at a density of 10 points per square meter, is used to create an elevation model, digital terrain model, and digital surface model of a forest. To identify individual trees, the DSM is segmented using the Euclidean distance of points as a clustering metric. Local maxima of the DSM are selected as centroids, while points one meter from the digital terrain model are ignored to avoid including ground points. From each cluster, geometric properties of individual trees are derived, including height, position, and crown diameter. Using these properties, the forest could be reconstructed with simple tree models. While useful for giving a general representation of a forest, the method's derived heights are generally underestimated, and diameters show no reliability in accuracy. Furthermore, it does not generalize to trees that do not have one clear peak, such as deciduous trees.

2.2 Skeletal Reconstruction

Ferley and Cani-Gascuel (1997) present a method for constructing a geometric skeleton from an unordered set of data points on a surface. When a Voronoi graph is created from the points, the skeleton is defined as the edges that are completely inside

the object. Because the object's surface is assumed to be unknown, it is found using a heuristic. The Delaunay tetrahedrization is derived from the Voronoi graph. The tetrahedra are merged until there are only two sets: one interior and one exterior. Voronoi edges that intersect the interior tetrahedron make up the skeleton (see Figure 2.1). To reduce noise, vertices are removed if their Delaunay volumes are too low. This information is used to create an implicit surface.

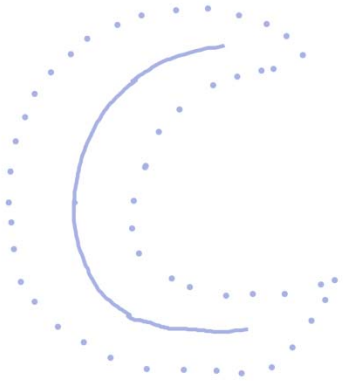

Figure 2.1- A point set with its derived skeleton.

A different approach is used by Verroust and Lazarus (1999) to solve this problem. An additional requirement is that a source point be defined as the root of the skeletal tree. The first step of the algorithm is building a neighborhood graph of the input points; each edge goes from a point to its nearest neighbor. Next, a geodesic graph is built by finding the shortest paths and distances between each vertex and the source point. Level lines are generated from a distance graph, each line consisting of

points that are equidistant from the source. The centroid of each level line is also computed. Because this method is designed for surface reconstruction, it is not an ideal choice for finding a skeleton in a LIDAR scan of a tree, which includes both surface points and disconnected leaf points.

Cornea et al. (2005) develop a method for extracting skeletons from geometric objects or point sets using a repulsive force function. Boundary voxels are assigned charges, and the net influence of all charges is assigned to each interior voxel. This creates a vector field within the object. Areas of the vector field where the magnitudes reach zero are of primary interest in creating the skeleton. Paths are seeded at saddle-points, while a force-following algorithm determines their directions. The discretization of these paths creates the object's core skeleton. For the first level skeleton hierarchy, points of the graph that qualify as sinks (vectors are mostly pointing inward) are also used to seed paths. The second level skeleton hierarchy can be created by seeding paths at areas of high curvature, such as corners. The user can specify the curvature that is required for seeding in the higher level hierarchies, which may have different requirements for different objects. Some drawbacks to this method are its potential for large memory requirements, and noisy results when the input is a scattered data set, as opposed to a geometric object.

In the CAMPINO algorithm (Bucksch & Lindenbergh, 2008), an octree-graph is generated from the point cloud. It is iteratively subdivided based on intersections with individual points. Once the octree has been obtained, a graph is generated consisting of M and T vertices. M vertices are created at the center of octree cells, while T vertices

are created in the middle of a cell's side. Next, vertices are merged to remove cycles. Merging is performed with the restriction that each M vertex must be connected to two T vertices. Finally, if an M vertex is connected to too many T vertices, the vertices are split. The resulting graph represents a tree's skeleton, and can be generated in O(n) time, where n is the number of points. In the SkelTre algorithm (Bucksch, Lindenbergh, & Menenti, 2009), the step for deriving a graph from the octree is optimized. The resulting skeleton can be derived in O(V) time, where V is the number of vertices created from the octree. Bucksch et al. (2009) show that the average distance of a point from a skeleton segment is smaller when using SkelTre as opposed to CAMPINO. The resulting skeletons are resilient to noise while giving good representations of trees. However, connectivity is not always maintained for thinner branches, and sometimes incorrect connectivity will exist. In addition, as mentioned by Livny, Yan, Olson, Chen, Zhang, and El-Sana (2010), both methods use pre-defined resolutions. This gives unreliable results because point density typically varies across the tree.

2.3 Tree Reconstruction

The problem of tree reconstruction is divided into two subproblems by Pfeifer et al. (2004) and Gorte and Pfeifer (2004). The first process, described by Gorte and Pfeifer (2004), derives connectivity information of the point cloud's underlying skeleton. To this end, a voxelization of the point cloud is created, and 3D neighborhood operations are performed to reduce noise and repair occlusion gaps. The voxel representation is reduced to a skeleton by removing voxels that do not alter connectivity. Dijkstra's

algorithm is performed to find a minimum spanning tree from the skeleton. A new segment is created for each branch in the tree, and lastly, points in the cloud are assigned to their nearest segment. The second process, described by Pfeifer et al. (2004), approximates the bare tree by fitting cylinders to each segment. For the points in a segment, normal vectors are estimated by fitting a plane to each point and its nearest neighbors (if a point lacks sufficient nearest neighbors, it is discarded as noise). The normal directions determine an initial guess for the axis direction. An initial guess for the axis location is taken from the center of gravity of points in a segment. Starting from these initial values, least squares adjustment is used to find a best fit for the cylinder in the data segment. Figure 2.2 shows one such cylinder. To approximate the next cylinder in a branch, the method uses cylinder following. The cylinder is shifted forward and uses the new points to re-adjust its parameters. Once this process is complete, the tree reconstruction consists of many disjoint cylinders. To create a smooth model, a new parameterization is created, using cylinder endpoints as observations. Smooth connections are made between cylinders, assuming the radius changes linearly. Unfortunately, a voxel representation may have unattainable time and memory requirements, depending on the size and complexity of the point cloud. Furthermore, according to Bucksch and Lindenbergh (2008), “thinning methods are known to be sensitive to changes on the object boundary. This sensitivity occurs, because the thinning methods try to approximate the medial axis, which in itself is already sensitive to small changes on the object boundary" (p. 117).

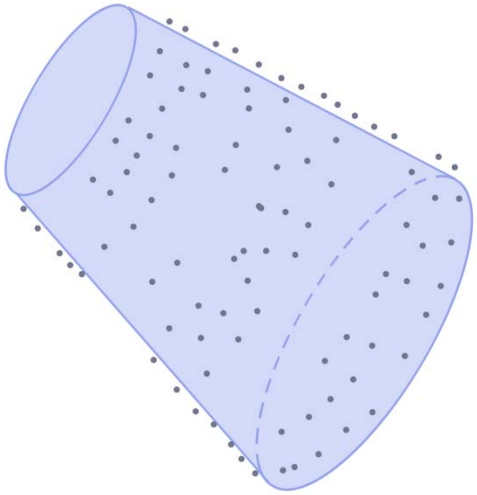

Figure 2.2 - A generalized cylinder with its corresponding points.

Xu et al. (2007) present a method for reconstructing trees that attempts to fill in occluded sections of the tree. The main tree skeleton is first created by connecting each point in the point cloud to all other points in the local neighborhood. Each edge is weighted based on its length. The shortest path is computed from all points to the root, and points are clustered based on their distance to the root and their adjacency in this graph. The centroid of each cluster becomes a node in the skeleton. To connect subgraphs that may be disconnected due to occlusion, a breadth-first search of this skeleton is performed. For each point below two-thirds of the tree height, a 45 degree cone is projected upward. If a subgraph G is found within a certain distance, it can be connected. The skeleton is then extended to be able to support the crown, which consists of all disconnected points. To enhance realism of the resulting model, skeletal

thickness is determined by allometry rules. The shape of the resulting model is influenced by properties of the tree species, requiring a library of tree parameters.

The reconstruction of a tree model from multiple photographs is explored by Neubert, Franken, and Deussen (2007). The shape of the tree is extracted from the photograph, and an attractor graph is created over expected branch paths using particle simulation. Density estimates are taken by back-projecting a point in 3D space to the input photographs, and used to create a voxel-based representation. The attractor graph is used to create a direction vector field over the voxels. Points are randomly placed in the direction field. Their paths, influenced by the direction vectors, are recorded to give an estimation of the tree's branches. Allometry rules are used to create geometry from the resulting paths. Although this method produces realistic results at a high level, its approach breaks down when simulating small twigs. In addition, it is unable to reproduce steep branching angles, because particles following such branches tend to merge together.

Once the shape of the crown is determined, synthesizing branches to support its entirety is more straightforward. Zhu et al. (2008) present an algorithm for reconstructing the tree's crown using alpha shapes. A shape is created out of the point set using Delaunay triangulation and the Lawson flip. First, a single tetrahedron is created that bounds the entire point set. Points are added one by one, and the shape is re-optimized at each step. If two tetrahedra don't meet the Delaunay property at an intermediate step, a new edge is added to correct this. Next, the radii of the circumspheres for each tetrahedron and tetrahedron face are computed. All tetrahedra

are classified as either external or internal by comparing their radii to a user-defined alpha value. If the radius is greater than alpha, the tetrahedron is exterior; otherwise, it is interior. Faces are classified according to the tetrahedra that they intersect. If a face is the intersection of one interior and one exterior tetrahedron, it is classified as a boundary face. The collection of boundary faces make up an approximation of the crown's hull. An appropriate alpha value must be somewhere between the smallest and largest tetrahedron radii in order to get a manifold crown mesh.

Côté et al. (2009) present a method for reconstructing 3D forest landscapes from LIDAR scans. Points are analyzed based on their light intensity to determine whether they should be classified as belonging to the tree's wood or its foliage. The method defined by Verroust and Lazarus (1999) is used to build skeletal curves from the wood points. Some of the foliage points are labeled as attractors. If an attractor is within a certain distance from a branch, it generates twigs from the branch to the attractor. The tree is filled by adding shoots according to the following rules: every branch tip can support foliage, and new shoots are added to segments with more available light. This method produces quality results even when the tree is underrepresented in scan points, but it requires user knowledge about the tree's characteristics.

The method developed by Livny et al. (2011) encodes scan data into a lobe-based representation, which consists of skeletal geometry and a set of alpha shapes that represent one or more parts of the canopy. The method makes use of a species library, where each species of tree has unique parameters such as shape, density, branching angle, and texture information. An input data set is classified as belonging to one of the

tree species, determining the parameters that will be used to build the representation. A skeletal shape is generated from the input data by a process of global optimizations, as detailed by Livny et al. (2010). The first species parameter determines the weighted values used when creating a shortest-path tree from points to the root. The second parameter determines how much thickness in a branch's radius decreases traveling from the root. The third parameter determines the size of texture lobes. Each lobe is populated with texture patches to fully synthesis the tree. Branches are procedurally generated using L systems for each species of tree. Large branches have anchor spots where other, smaller branches can be docked at predefined angles, in order to control the shape and fullness of the foliage. The shape must fit in the size of its texture lobe, and the fullness has to follow the application's level of detail rules. This method produces realistic trees, although it has limited use for trees with dense foliage or canopies that have long, sparse twigs.

2.4 User Assisted Plant Synthesis

The generation of 3D plant models from user sketches is explored by Chen, Neubert, Xu, Deussen, and Kang (2008). From a 2D sketch of the skeleton, the procedure first searches a library of tree templates for a model with a projection that closely matches the characteristics of the sketch. Branches are reconstructed by converting the sketch to a Markov tree. The depth of a segment's endpoint is jointly determined by the parent segment's angle and by the template's parameters. Branching angles are iteratively optimized to achieve a more realistic appearance. The tree is

populated with small branches by randomly copying and pasting sections of the base skeleton, and then scaling and orienting them according to the template's parameters. Lastly, the crown is populated with leaves taken from the template. The benefit of this method is it can produce a realistic-looking tree with only a few strokes. However, it produces unnatural trees for some sketches due to the branch angle optimization step. In addition, its branch propagation stage fails for tree species that do not have the property of self-similarity.

2.5 Surface Comparison

With the problem of reconstructing an object, it is useful to have a way of analyzing the new object's accuracy. The Metro program, developed by Cignoni, Rocchini, and Scopigno (1998), approaches this problem by using the Hausdorff distance, as defined in Section 1.8. A user-defined number of samples are taken on a surface S_1, and for each, it calculates the closest distance to the second surface, S_2. The reported statistics include the maximum error of all samples, the mean error, and the root mean square error. All values are given as both absolutes and percentages of the bounding box diagonal of S_1. To ensure precise results, it is reported that the sample point density should be at most 0.1% of the bounding box diagonal. The Mesh program, developed by Aspert, Santa-Cruz, and Ebrahimi (2002), is also based on the Hausdorff distance, but showcases increased speed and reduced memory usage. Here, it was found that point densities below 0.5% of the bounding box diagonal give the most stable results. The MeshLab program, developed by Cignoni, Corsini, and Ranzuglia (2008), incorporates a

Hausdorff filter as part of its suite of mesh processing tools. It includes the option of colorizing a surface S_1 based on the distance of each sample point from the surface S_2.

2.6 Conclusions

The methods of Pyysalo and Hyyppa (2002), Holmgren and Persson (2004), and Morsdorf et al. (2004) are well-suited for recognizing tree structures from range scans, but a more careful analysis is required for reconstructing individual trees. The key issues in this problem are overcoming self-occlusion (by leaves and small branches) and distinguishing surface points from leaves. Many existing methods for deriving the skeleton of an object are highly sensitive to noise, such as those developed by Ferley and Cani-Gascuel (1997), and Verroust and Lazarus (1999). The problem of self-occlusion is generally addressed by using available information to estimate missing data (Pfeifer et al., 2004) and patching together segmented results (Xu et al., 2007). To make up for the unorganized nature of point clouds, some methods utilize pre-existing information about the tree's species, such as those presented by Xu et al. (2007), Côté et al. (2009), and Livny et al. (2011). Leaves and twigs can be procedurally generated to account for undersampling of the crown, as seen in the methods of Xu et al. (2007), Neubert et al. (2007), and Livny et al. (2011).

The method of this research is more concerned with accurate reconstruction than it is about visually pleasing results, meaning leaves are not synthesized. User sketch lines guide the location of important segments to help overcome the problems of disconnectedness and self-occlusion. Segment radii are estimated using RANSAC,

because it is well-known for its resilience to noise. The accuracy of the method is calculated using the Hausdorff filter of the MeshLab program (Cignoni e al. 2008), which measures the distance between two surfaces at a multitude of points.

2.7 Summary

This chapter summarized existing literature on the reconstruction of trees from point clouds. Different approaches for extracting tree properties, skeletal structures, and crown shapes were examined. The inclusion of a surface accuracy metric is addressed. Finally, conclusions were presented on the state of research in the field.

CHAPTER 3. METHODOLOGY

This chapter gives a summary of the central method of this research. Details of the research framework are given, including variables and hypotheses. The chapter concludes with an overview of testing tools.

3.1 Procedure

This section describes the procedure used to generate output models, outlined in Figure 3.1.

3.1.1. Input

Two types of input data were available for this research: pre-existing point clouds taken from actual LIDAR scans, and synthetic tree meshes. The second type needs an additional preprocessing step in order to produce point clouds. Rays were cast toward the center of the model from evenly spaced points on the surface of a sphere, and the nearest intersection coordinate for an individual ray is added to the point cloud. The user determines how many rays are used. For accurate estimation, most objects require over a thousand points.

3.1.2 Sketching

The input points are displayed in a window to the user, where he or she can manually or automatically create a sketch overlay. A sketch in this context consists of a hierarchy of nodes connected with lines. For reconstruction purposes, each node is projected into 3D space. Using the current projection, view, and world matrices, two 3D vectors are computed from the node's 2D location: one pointing to the near clip plane, the other pointing to the far clip plane. Subtracting these two vectors yields the node's 3D projection.

3.1.3 Reconstruction

As soon as a sketch exists, the user can begin the reconstruction process. Beginning from the bottom of the sketch tree, a thin slice of the point cloud is taken, perpendicular to the current sketch line and parallel to the projected node line (the thickness of this slice is determined by a static epsilon value). Only the points in this slice are used to represent the node. To further refine the results, the user can choose to ignore points that are too far from the projected node line. The resulting collection of points is passed to the RANSAC function, where the optimal radius and origin are calculated. Thus, each sketch node generates a circle that closely matches a slice of the tree skeleton. If not enough points are found by RANSAC to approximate a circle, that node is skipped for the remainder of the procedure.

3.1.4. Output

The circles are discretized into triangles and connected to form polygonal generalized cylinders. If some limbs in the model appear inaccurate, the user can still edit the sampling size of individual nodes, or delete them entirely. The output mesh will be entirely reconstructed to reflect these changes. Once the user is satisfied with the results, he or she can export the mesh for the analysis stage.

3.2 User Role in Method

The inclusion of a sketch overlay came from the need of the user to control accuracy of the reconstruction. With fully automatic methods, most of the input points are used to find the skeleton, regardless of whether these points belong to the skeleton. On the other hand, a fully manual method would be extremely tedious and time-consuming. An ideal algorithm would incorporate as much automation as possible, while allowing the user to perform some manual adjustments to improve accuracy. Many of the automatic methods presented in chapter 2 incorporated some measure of user control, such as input parameters, due to the varied appearance of trees. The method used in this research gives significant control to the user in the form of sketches. The user can also control reconstruction by editing the sample size for each sketch node. If more automation is preferred, the user can opt to automatically generate a sketch, with evenly distributed nodes and maximal sampling sizes. This sketch can later be refined by the user. Therefore, the method fits in a small range of control between full automation and full manual, as seen in Figure 3.2.

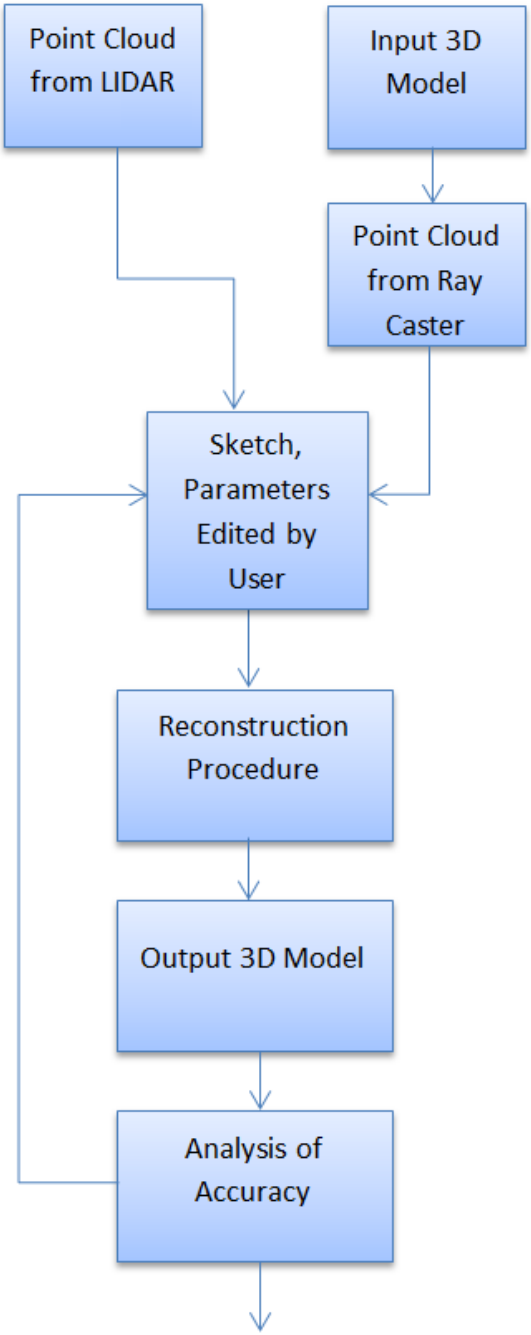

Figure 3.1 - Overview of the reconstruction procedure.

Figure 3.2 – The method's range of automation.

3.3 Testing Methodology

Once the reconstructed model appears acceptable to the user, it can be exported for analysis. Both the original and the reconstructed models are imported into MeshLab, an open-source utility that allows the simple processing of meshes (Cignoni et al., 2008). Its Hausdorff distance tool (seen in figure 3.3), reports on the similarity between two meshes. The approach is to take a predetermined number of samples on one surface, and for each sample, find the nearest point on the second surface. The minimum distance between the surfaces provides a local metric for accuracy. The numerical results given by MeshLab include the maximal and mean square errors, given as both absolute values and percent relative to the diagonal of the mesh's bounding box. Only the percent values are reported in this research. The Hausdorff distance filter is applied twice for each model, giving both forward and backward analyses. The forward analysis samples the input model and finds its nearest distance to the output model, while the backward analysis does the opposite. This gives a more thorough representation of accuracy than a one-way analysis.

3.4 Hypotheses

The following hypotheses were tested:

- H_0: The method does not produce an acceptable approximation of the original model.
- H_a: The method does produce an acceptable approximation of the original model.

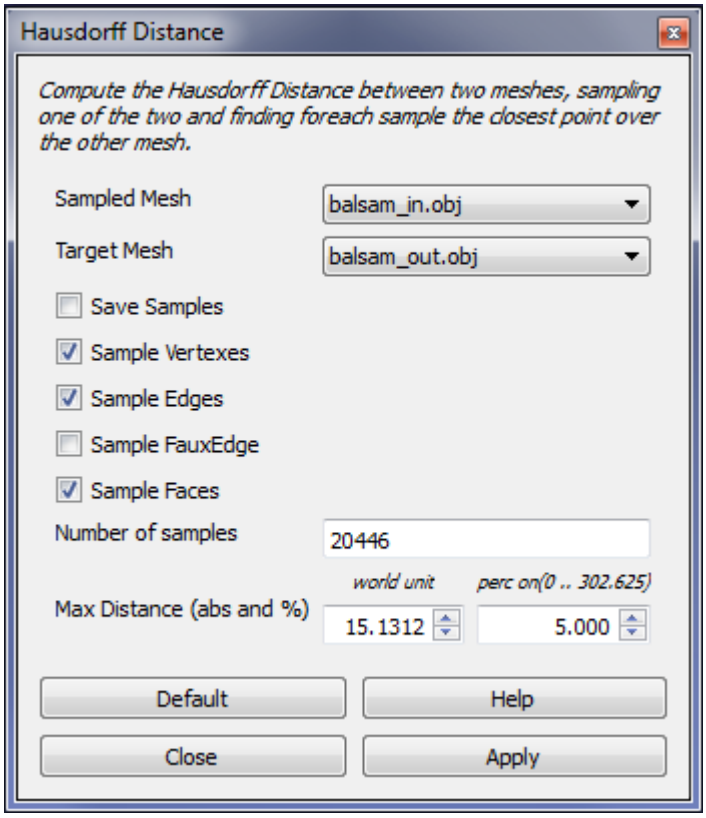

Figure 3.3 – The Hausdorff distance filter interface in MeshLab (Cignoni et al., 2008).

3.5 Variables

The independent variable is the number of nodes in the user sketch. The effect of this variable is measured on the root mean square error between the surfaces of the input and output meshes.

3.6 Assessment of Results

The input and output meshes are compared using the Hausdorff distance of their surfaces, which can be used to find the mean square error. Due to the need for surface comparison, only the artificial tree meshes are tested. To incorporate the effect of point cloud density on the results, each tree is analyzed at multiple sampling quantities. Both the user-guided and automatic approaches are tested in order to show how accuracy can improve with user assistance.

3.7 Tools and Environments

The following tools were used in this research. Development was done in C++, using Microsoft Visual Studio 2010 for debugging. Additional libraries included OpenGL, for drawing commands; the GLM library for matrix and vector calculations (Riccio, 2011); and AntTweakBar, for the user interface (Decaudin, 2011). Processing of input meshes was done in Blender 2.63. Analysis was performed in MeshLab v.1.3.1, using its Hausdorff distance sampling filter (Cignoni et al., 2008).

Two separate machines were used for testing. The following are the specifications for the first machine.

- Dell Precision T7500
- Microsoft Windows 7 Professional, 64 bit
- 12.0 GB RAM
- 2 x Intel Xeon E5520 @ 2.27 GHz each

- NVIDIA Tesla C1060 & NVIDIA GeForce GTX 480

The following are the specifications for the second machine:

- Customized CyberPower desktop PC
- Microsoft Windows 7 Professional, 64 bit
- 4.0 GB RAM
- 4 x Intel Core i5-2400 @ 3.10 GHz each
- NVIDIA GeForce GTX 460

3.8 Summary

This chapter began by explaining the reconstruction algorithm, as well as the role of the user in the method. The hypotheses and variables were given, and the quasi-experimental nature of this research was described. Finally, the details of the testing tools were given.

CHAPTER 4. IMPLEMENTATION

This chapter gives the implementation details of the method described in chapter three. It begins with an overview of classes in the system architecture. This is followed by sections on each module unique to this research.

4.1 System Architecture

The implementation was built on top of a graphics engine developed at Purdue's High Performance Computer Graphics laboratory (Vanek, 2011). The engine provided basic functionality for this research, including a real-time rendering display, trackball interface, and various rendering options. Figure 4.1 shows an example of the implementation's interface.

The Main file is the entry point for the running program. It handles user input, file selection, and scene creation. After the user selects a file name, it is used to initialize the Scene object, which loads and renders the file into a wireframe mesh. The Scene class contains two more important objects: instances of the Sketch Tree and Point Cloud classes. The Point Cloud class loads points related to the current tree from file and displays them; or, if no such file exists, it calls on the Ray Caster class to create one. The SketchTree class contains a linked list of Sketch Nodes.

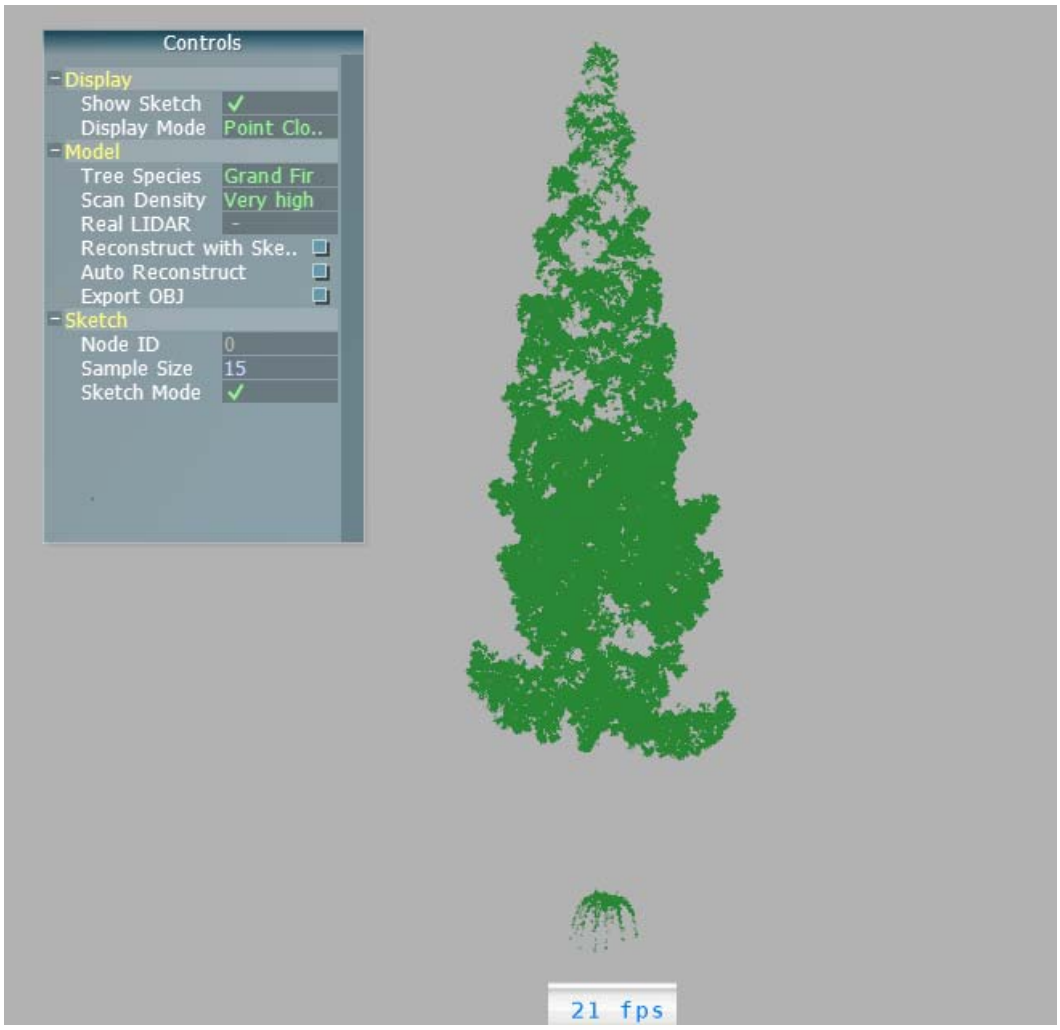

Figure 4.1 – The implementation's interface.

Each instance of Sketch Node handles the projection of itself into 3D space, as well as RANSAC estimation of the radius and origin. It returns an array of triangles that make up the tree limb slice to the SketchTree object, where they are added to a triangle soup that makes up the reconstructed model. See Appendix A for more details on these data structures. An overview of the entire system is shown in Figure 4.2. The highlighted box represents the core engine.

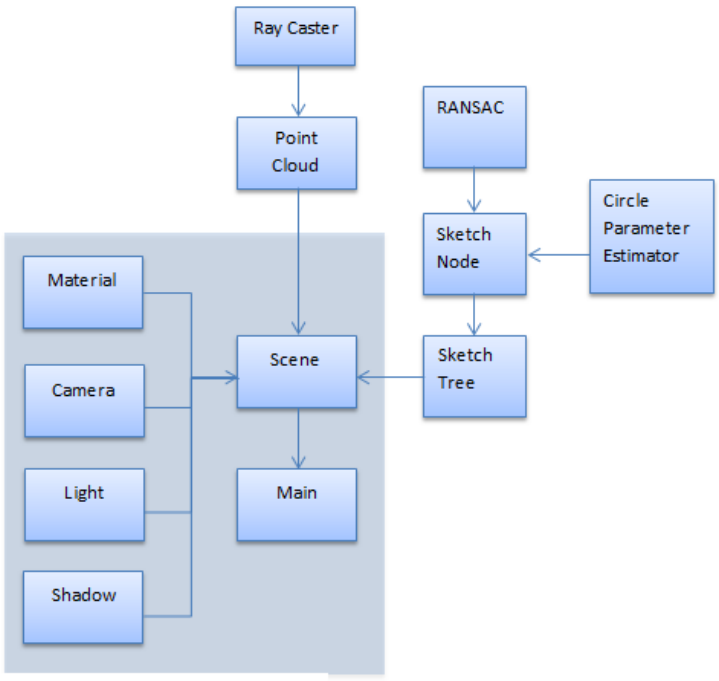

Figure 4.2 – The architecture of the implementation.

4.2 Test Data Generation

When using a 3D model as input, a ray caster is used to obtain the point cloud. Rays originate from the surface of a sphere and point roughly towards the center of the model, randomized by a small amount to avoid regularity. The exact origins of the rays are located on the vertices of a spherically subdivided convex hull. The number of subdivisions determines the point density of the resulting cloud; for the purposes of this research, it was found that between six and eight subdivisions (yielding between 73,728 and 1,179,648 data points) were sufficient. For each ray, intersections are detected by calculating the ray-triangle intersection of every facet in the model. The intersection point with the shortest distance to the ray origin is added to the point cloud. See

Appendix B for the implementation. After calculations are finished, the point cloud is saved to file for future use.

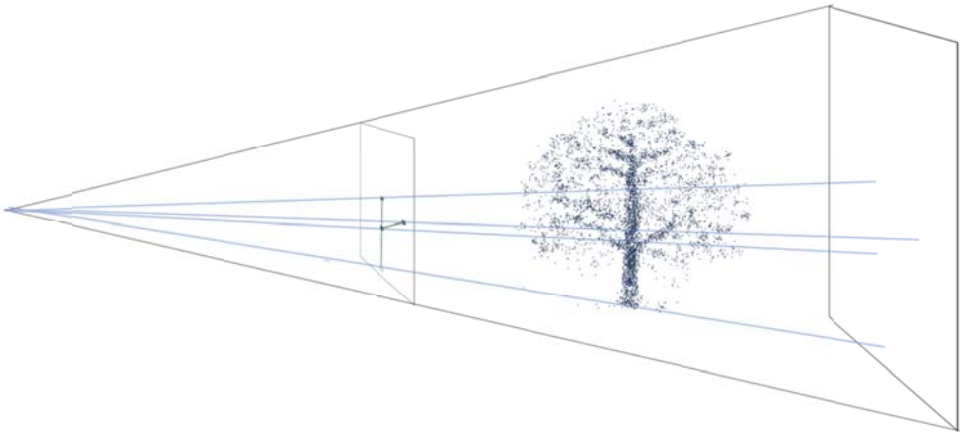

Figure 4.3 – An overview of a scene with a sketch and a point cloud.

4.3 Sketching

Each sketch node contains its 2D screen position and pointers to its parent and child nodes. An important step in the reconstruction process is deriving a 3D vector, *A,* from this 2D point using GLM's unproject function. As input, this procedure takes the current view, projection, and world matrices. *A* is found by unprojecting the 2D point to both the front and back clip planes and finding the difference between these two vectors. The cross product of this node's *A* and the parent node's *A* is again crossed with A to obtain the normal vector *N*. If the node has no parent, a default vector of (0, -1, 0) is used in this calculation instead. A diagram of the sketching environment is illustrated in Figure 4.3. The rectangle in the center is the viewing screen with a sketch, and the light blue lines are the projections of the sketch into 3D.

4.4 Reconstruction

Before reconstruction can occur, the SketchTree object calls a method called CollectPoints in each SketchNode, passing a reference to the point cloud. For each point, the method calculates the point-plane distance to the node's plane (represented by the projected line and its calculated normal). If the distance is less than a constant epsilon value, it is added to a collection called nearPts. This function also calls itself for each of the node's children, if they exist. When reconstruction occurs, the SketchTree object recursively calls the CalcDisc function for each SketchNode object. This function has its own collection called testPts, a subset of nearPts that is used in the RANSAC function. It is populated by checking the point-line distance of each element in nearPts to the projected node line. If a point's distance is less than the user-defined lineEpsilon, it is added. Here, the RANSAC function is called. It takes four parameters: an array of values to fill (in this case, the center and radius of the circle), a function for calculating those values (in this case, a function called CircleParamEstimator), the collection of test points, and the desired probability for no outliers (a constant double with the value 0.999). If RANSAC could not find an estimate, lineEpsilon is increased and the process begins again. This continues until either an estimate is found or testPts equals nearPts. See Appendix B for the implementation.

Two additional functions are required before a node is fully reconstructed. First, the disc is projected to the node's plane, using its normal. Next, triangles must be generated for the mesh. Generally, this is done by creating triangles from the disc to the

parent's disc. If the node either has no parent or child nodes, the result is a triangle fan around the disc. These triangles are added to the entire mesh's triangle soup.

If the user chooses to reconstruct again, the aforementioned data structures are cleared, and the process begins from scratch.

4.5 Analysis

Analysis of the method is performed using MeshLab, a tool that implements a Hausdorff Distance filter (Cignoni et al. 2008). Two OBJs are imported as separate layers: the original tree skeleton and the reconstruction. Because the reconstruction is only concerned with simulating the tree's main skeleton, leaves and small twigs are removed from the original before analysis. First, the backward analysis is calculated. The filter randomly selects points on the reconstruction's vertices, edges, and faces, and calculates their nearest distances to the original. The number of samples is randomized, although it is initially based on the number of vertices in the original model. The maximum acceptable distance is automatically selected based on the length of the original mesh's bounding box diagonal. The output of this filter includes the minimum, maximum, mean, and root mean square of the distances. This process is repeated for the forward analysis by selecting points on the original's surface and calculating their nearest distances to the reconstruction.

To visualize the results, a colorize filter is applied by vertex quality. The range of colors is based on the maximum and minimum calculated distances. Points with values

below the minimum will be colored red, values above the maximum will be colored blue, and values in between are interpolated.

4.6 Summary

This chapter began with an overview of the implementation's architecture. It also went over the details of three important procedures: preprocessing, sketching, and reconstruction. The chapter concluded with a description of how the method's output is analyzed using a third-party tool.

CHAPTER 5. RESULTS AND ANALYSIS

This chapter presents the results obtained from the method. The results on real LIDAR scans are presented first, followed by analysis on synthetic trees. The chapter concludes with a summary and discussion.

5.1 Results on Real Trees

The method was performed on two LIDAR scans to see how it would work on data obtained from real trees. The first example is shown in figure 5.1. The second is seen in figure 5.2.

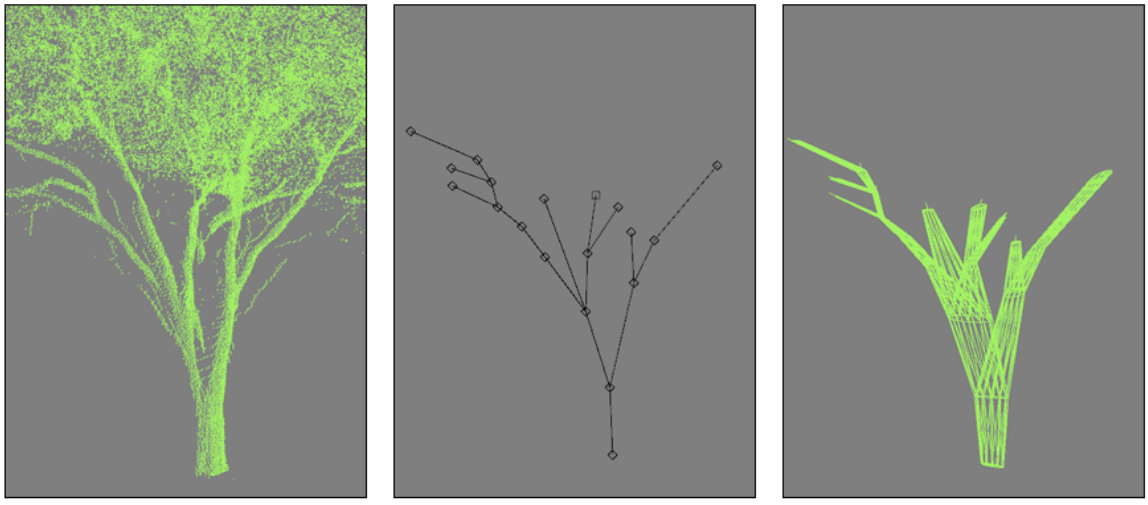

Figure 5.1 – The point cloud, user sketch, and output mesh of the first LIDAR scan.

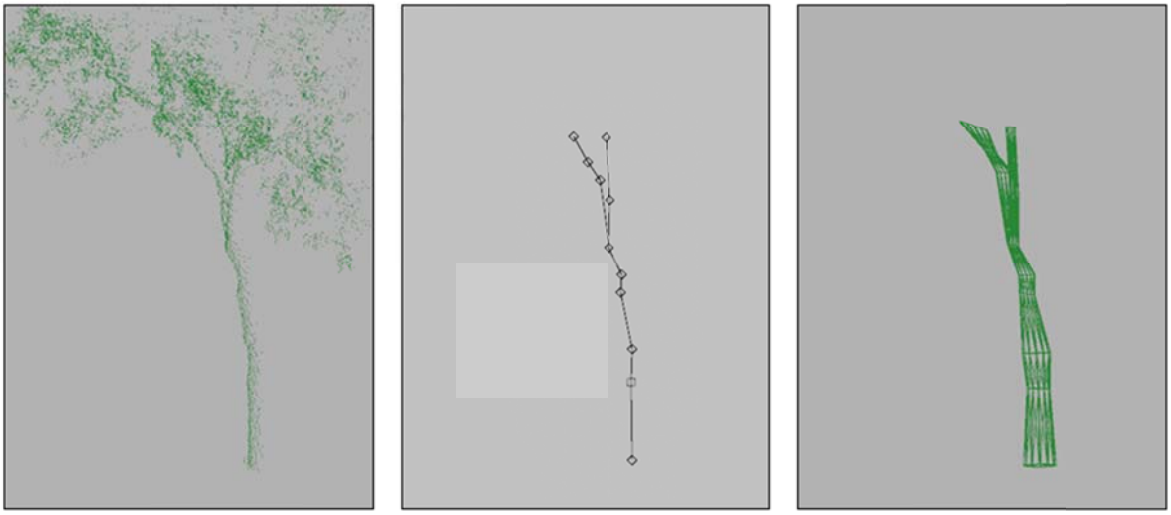

Figure 5.2 – The point cloud, user sketch, and output mesh of the second LIDAR scan.

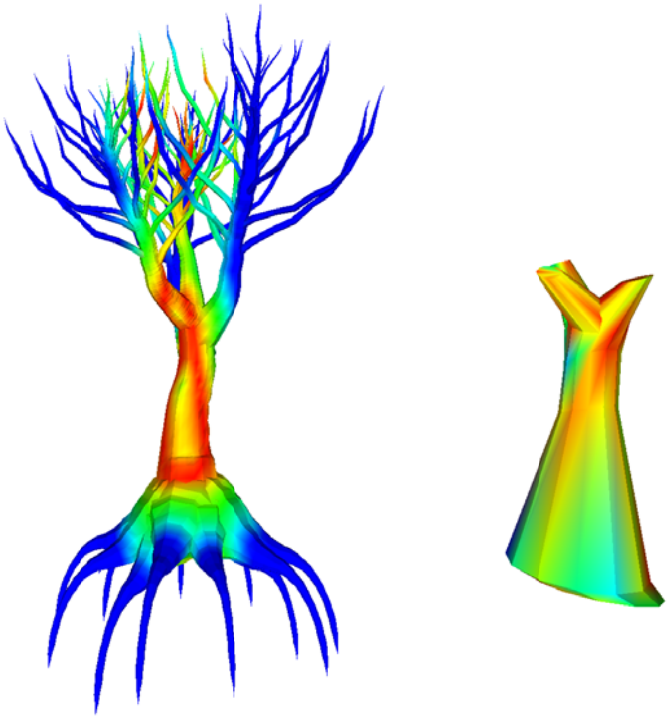

Figure 5.3 – The Bristlecone Pine input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

5.2 Results on Synthetic Trees

For analysis, the following synthetic models were used: Bristlecone Pine, Coast Live Oak, Little Walnut, Giant Sequoia, Monterey Cypress, Grand Fir, Balsam Poplar, Oregon Ash, and Vine Maple. All point clouds had approximately 294,000 sample points; the ray caster created 294,912 rays, but some rays found no intersection points. The Hausdorff distance filter was applied as both original to reconstruction (forward analysis) and reconstruction to original (backward analysis). In the figures that follow, original models are colorized based on the results of forward analysis, while reconstructed models are colorized based on backward analysis.

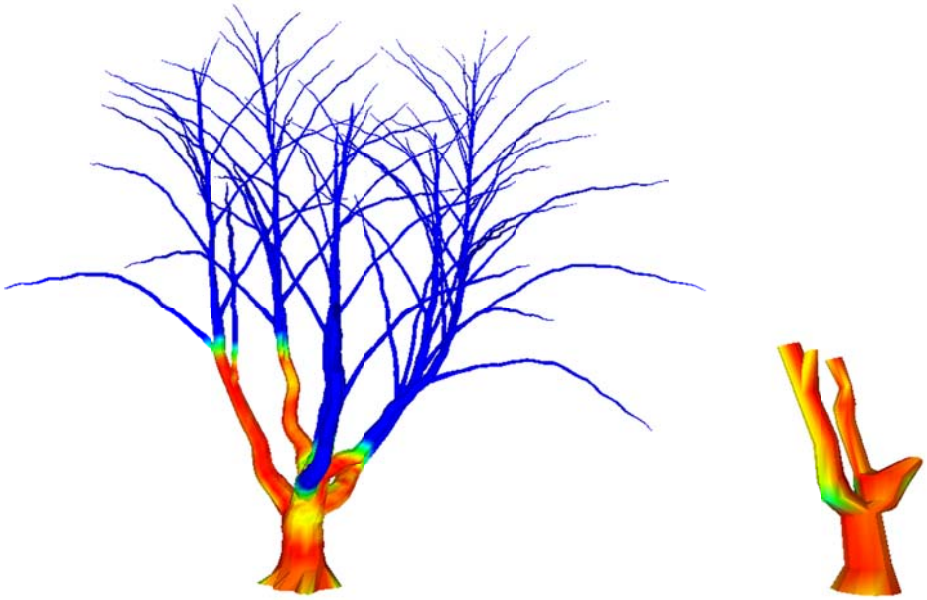

Figure 5.4 – The Coast Live Oak input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

The first tree to be analyzed was a Bristlecone Pine model. This tree is a strong example of self-occlusion (see figure C.2). Because of the broad crown and roots of the

input mesh, the point cloud was separated into three distinct groups: the crown, the roots, and the very center of the trunk. It was necessary to space the sketch nodes across the gaps. Figure 5.3 shows the input and output tree skeletons. It is apparent that the sampling gap near the base negatively affected the reconstruction. Its trunk slopes linearly, unlike the crooked shape of the original. The central part of the tree is more closely represented, although there is still error due to scarcity of sampling points.

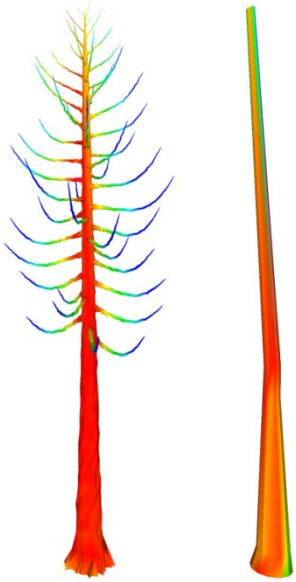

Figure 5.5 – The Giant Sequoia input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

With the Coast Live Oak, seen in figure 5.4, the numerous branches arranged in a circular fashion made it difficult to find an ideal perspective from which to sketch (see figure C.3). Also, because some of the branches were so close together, there was an issue where nodes of adjacent branches would merge their results together. The most inaccurate section of the output model was the center of the trunk. Here, the multitude

of branches coming out at different angles could not be captured accurately using this method.

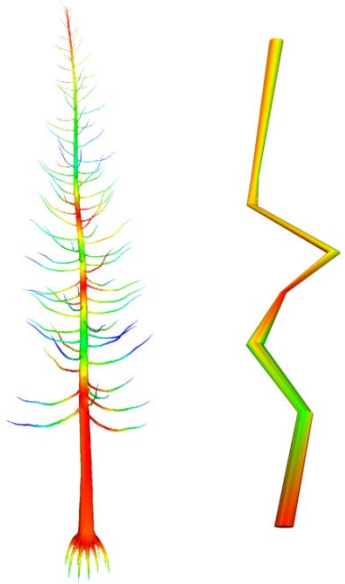

Figure 5.6 – The Grand Fir input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

Coniferous tree species proved to be the most difficult to reconstruct with this method, due to their high density foliage and undersampled trunks. For the Giant Sequoia, seen in figures 5.5 and C.4, this was avoided by sampling only the highest density sections on the trunk, as well as the tip of the tree. For the Grand Fir, seen in figure 5.6 and C.5, an attempt was made to sample the trunk between spaces in the foliage. However, the noise from the foliage produced inaccurate locations for the disc centers.

Two trees featuring thin, undersampled trunks were the Balsam Poplar and Little Walnut. In the case of the Balsam Poplar, the reconstruction followed the curve of the

trunk, but not its thickness, as seen in figure 5.7. Its sketch is seen in figure C.1. The Little Walnut's trunk was more easily constructed. In both cases, the densely-packed branches were the most inaccurate features of the reconstruction. The Little Walnut can be seen with several branches merged into one in figure 5.8. Its sketch is shown in figure C.6.

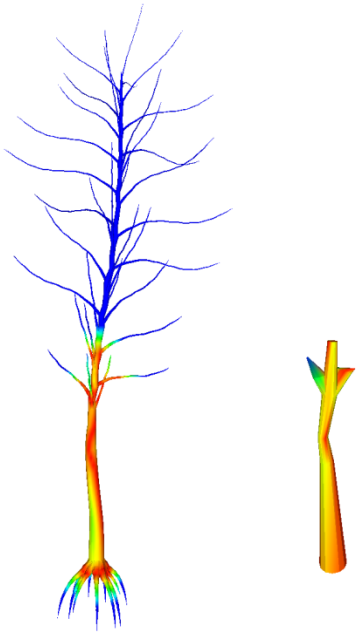

Figure 5.7– The Balsam Poplar input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

The Vine Maple, Oregon Ash, and Monterey Cypress all had distinctly shaped trunks that were generally well-captured. The size of the tree limbs did not follow allometry rules, however. This can be seen in the center of the Vine Maple (figures 5.9 and C.9), and near the base of the Monterey Cypress (figures 5.10 and C.7). The thickness of the Oregon Ash, as seen in figures 5.11 and C.8, was generally slightly off, with the exception of the highly inaccurate leftmost branch.

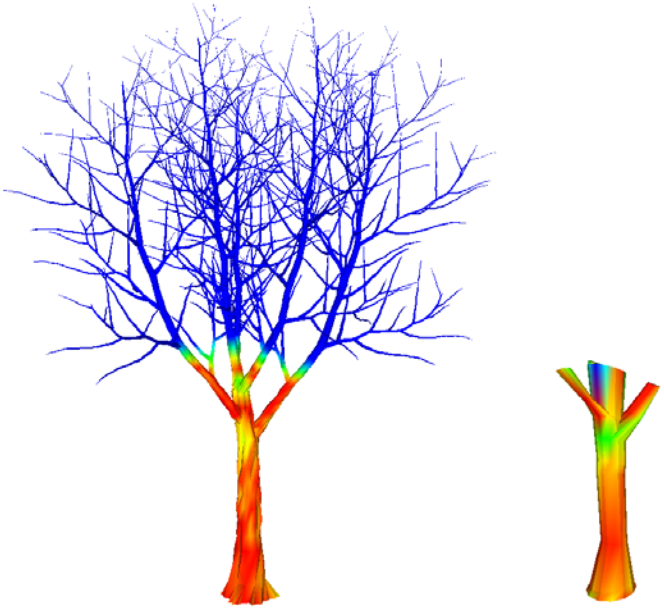

Figure 5.8 – The Little Walnut input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

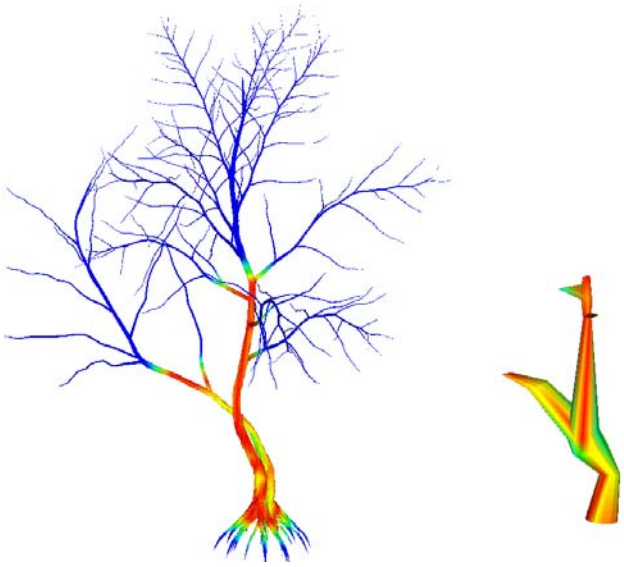

Figure 5.9 – The Vine Maple input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

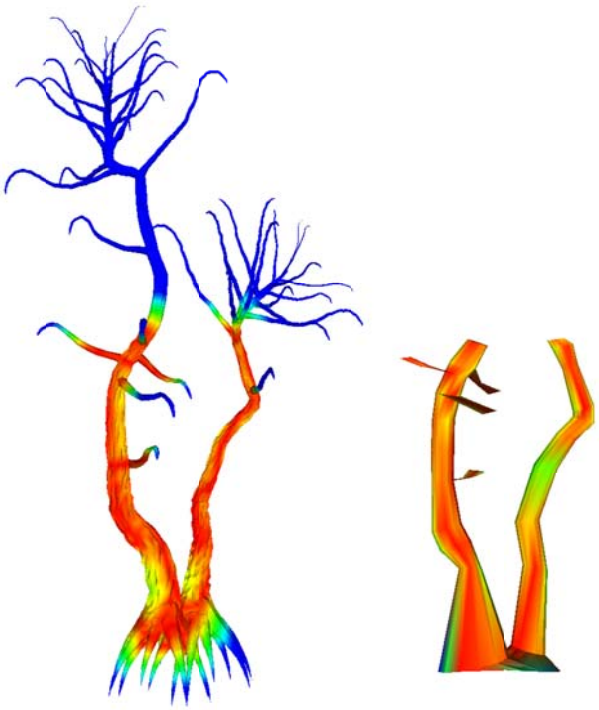

Figure 5.10 – The Monterey Cypress input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

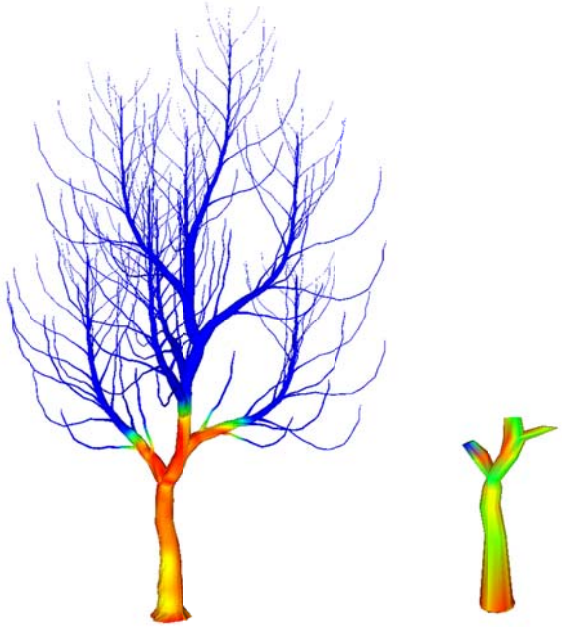

Figure 5.11 – The Oregon Ash input (left) and output (right) meshes, rendered in MeshLab (Cignoni et al., 2008).

5.3 User Sketch Results

Table 5.1 gives a summary of the results, grouped by category. The Backward Analysis group contains the results with the output mesh as the sampled object, and the input mesh as the target. Forward Analysis reverses the roles of these two objects. The maximum distance, mean distance, and root mean square error are all given as percentages of the sampled mesh's bounding box diagonal.

It was expected that the forward analysis would show higher error values, due to the distance that unrepresented parts of the skeleton would have to travel. This was not always the case. In fact, five of the trees tested (Coast Live Oak, Little Walnut, Balsam Poplar, Oregon Ash, and Vine Maple) had higher errors with the backward analysis. This is presented in figure 5.12.

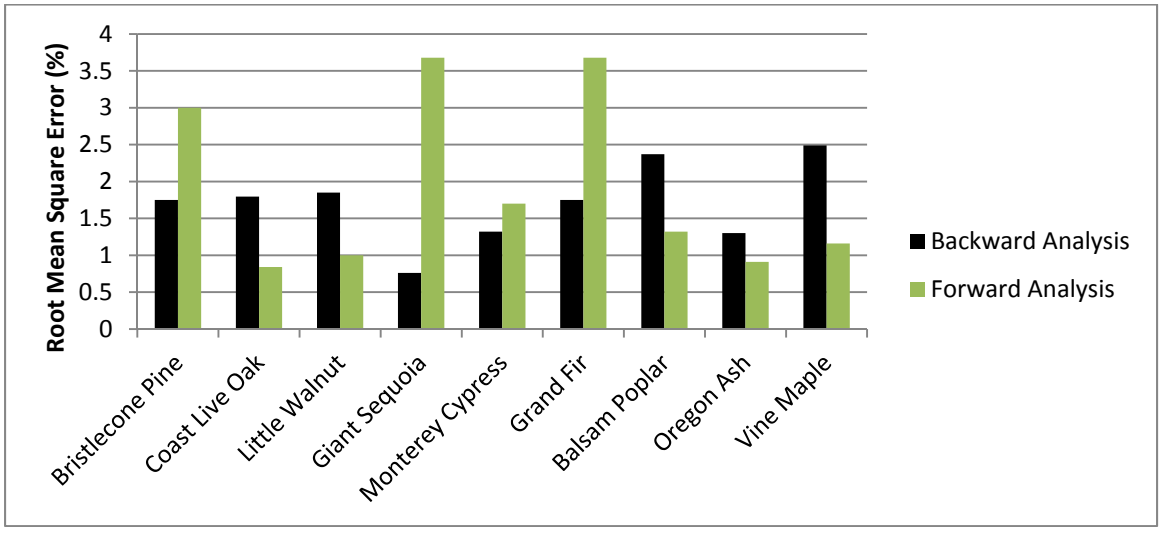

Figure 5.12 –Backward and Forward Analysis errors.

5.4 Automatic Sketch Results

In addition to user-guided sketches, automatic sketches were used to create reconstructions of each tree. With this approach, sketch nodes were placed at regular intervals from top to bottom of the tree display. The sampling size for each sketch node was set at the maximum value. Essentially, this uses the entire point cloud for the reconstruction of a mesh. The Hausdorff distance filter was applied to these objects as well. The results are shown in Table 5.2. A comparison of the root mean square error between the user sketch results and the automatic sketch results is shown in figure 5.13. On average, the error of the user-guided approach was 56.4% lower than the automatic.

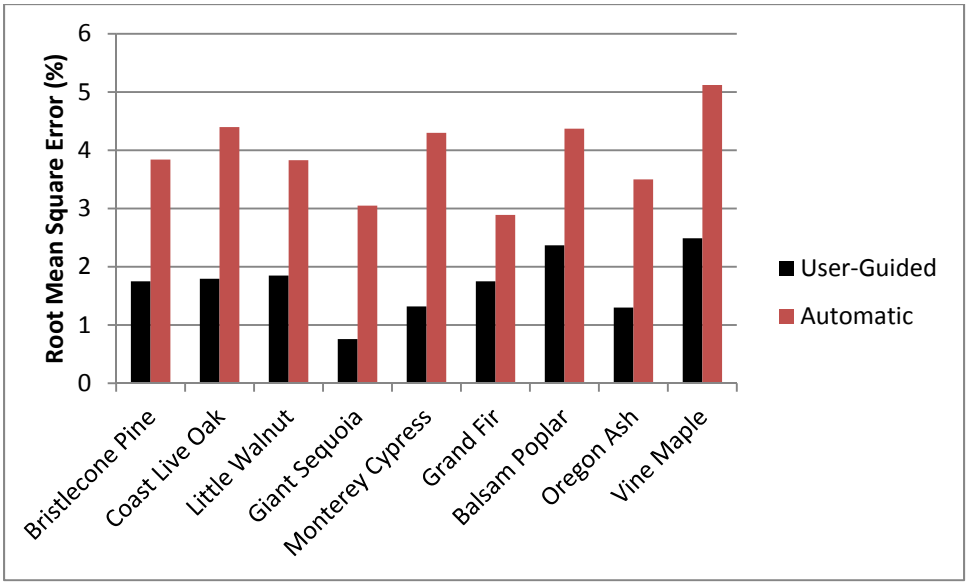

Figure 5.13 – Backward analysis error of user-guided and automatic reconstruction.

5.4 Effect of Sketch Complexity on Accuracy

In the user-guided process, each tree was given a different number of sketch nodes. This was a side effect of trying to produce the most accurate mesh possible for

each tree. The correlation between root mean square error and sketch node count was found to be -0.08 (see figure 5.14). To further explore this topic, a single tree limb of the Monterey Cypress model was reconstructed with two nodes, and again with four. The two node sketch resulted in a root mean square error of 5.12%, and the four node sketch resulted in 3.37%. These differences are shown in figure 5.15.

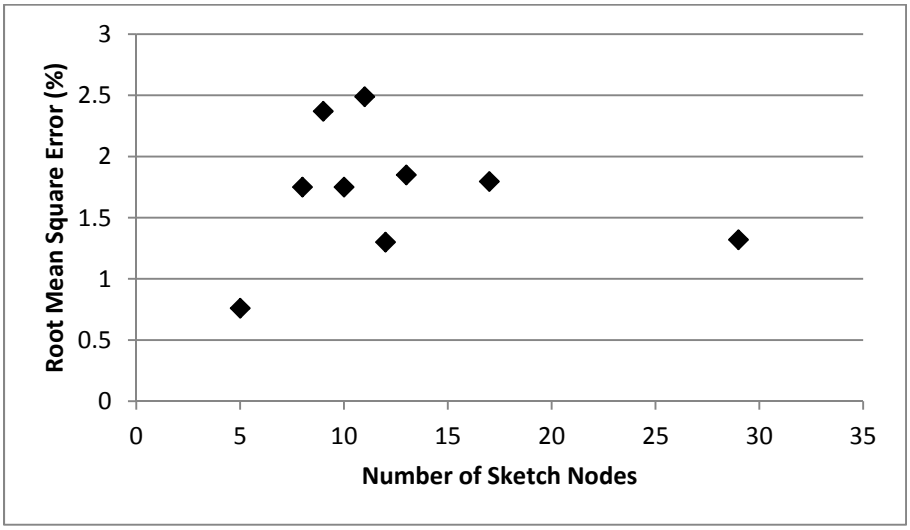

Figure 5.14 – The relationship between sketch node count and error in the user-guided reconstructions.

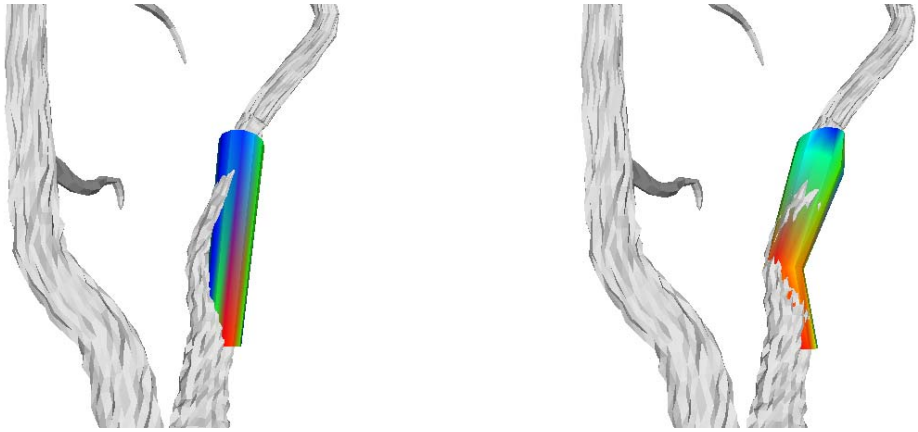

Figure 5.15 – A Monterey Cypress tree limb rebuilt using two sketch nodes (left) and four (right), rendered in MeshLab (Cignoni et al., 2008).

5.5 Summary

This chapter began with a presentation of individual synthetic and real tree results, including the visualization of the synthetic tree errors as they appeared in MeshLab. Also presented were tables that summarized the numerical results of both user-guided and automatic reconstructions. Lastly, the relationship between accuracy and sketch node count was discussed.

Table 5.1

User-Guided Sketch Analysis Results

	Bristlecone Pine	Coast Live Oak	Little Walnut	Giant Sequoia	Monterey Cypress	Grand Fir	Balsam Poplar	Oregon Ash	Vine Maple
Reconstruction									
Sketch Nodes	10	17	13	5	29	8	9	12	11
Reconstruct Time (ms)	6150	212	100	49	857	1003	148	100	89
Backward Analysis									
Samples	27125	42589	82590	33228	60253	59761	41647	41884	70668
Bounding Box Diagonal	77.13	100.38	62.69	541.88	136.05	370.6	111.05	68.99	39.34
Max. Distance (%)	5.69	9.03	7.02	2.91	5.95	4.68	7.78	6.02	7.85
Mean Distance (%)	1.31	1.25	1.47	0.53	0.94	1.45	1.77	1.09	1.91
Root Mean Square Error (%)	1.75	1.796	1.85	0.76	1.32	1.75	2.37	1.3	2.49
Forward Analysis									
Samples	24950	9502	12278	42935	50313	86790	21094	6805	29734
Bounding Box Diagonal	189.97	385.34	199.05	594.28	258.28	402.1	302.62	239.66	123.14
Max. Distance (%)	5.85	2.6	3.15	9.1	4.87	9.2	3.67	2.88	3.2
Mean Distance (%)	2.57	0.55	0.65	2.57	1.19	2.87	0.97	0.61	0.85
Root Mean Square Error (%)	3	0.84	1	3.68	1.7	3.68	1.32	0.91	1.16

Table 5.2

Automatic Sketch Analysis Results

	Bristlecone Pine	Coast Live Oak	Little Walnut	Giant Sequoia	Monterey Cypress	Grand Fir	Balsam Poplar	Oregon Ash	Vine Maple
Reconstruction									
Sketch Nodes	18	18	18	18	18	18	18	18	18
Reconstruct Time (ms)	21549	77845	39912	26418	20004	18844	37521	52896	32227
Backward Analysis									
Samples	26939	42008	77292	34007	60361	60274	42172	42124	61439
Bounding Box Diagonal	203.188	337.9	201.43	558.32	257.05	314.5	243.03	216.37	112.8
Max. Dist. (%)	9.35	11.4	9.88	9.96	10.05	7.46	9.92	11.07	10.92
Mean Dist. (%)	3.1	3.6	3.02	2.53	3.73	2.56	3.83	2.82	4.32
RMS (%)	3.84	4.4	3.83	3.05	4.3	2.89	4.37	3.5	5.12
Forward Analysis									
Samples	39501	56168	115150	48954	743608	69918	46885	59297	101164
Bounding Box Diagonal	189.97	385.34	199.05	594.26	274.8	402.1	302.62	239.66	123.14
Max. Dist. (%)	10.68	8.77	10.12	9.4	9.35	7.82	8.03	9.03	9.16
Mean Dist. (%)	3.33	2.33	2.73	2.38	2.41	2.4	2.64	2.78	2.34
RMS (%)	4.21	3	3.59	3.07	3.06	2.3	3.48	3.6	3.06

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This research has demonstrated the capability of using RANSAC with both automatic and user-guided methods to rebuild tree models from point clouds. The results of the previous section show that the user-guided reconstruction method is much more accurate than the automatic method. There was not any correlation between the number of sketch nodes and the accuracy of the resulting mesh, however. Instead, it was more important that the placement of nodes be highly selective. Figure 5.13 shows a situation where more nodes can improve on the accuracy. If an area has too much noise, more nodes can be detrimental to the accuracy. The number of nodes should be selected by the user on a case-by-case basis.

6.2 Limitation of Method

While the results of this research are promising, several limitations must be addressed.

- The Hausdorff distance is an accepted tool for measuring the difference between two meshes. According to Cignoni et al. (1998), however, it is not always reliable

when the meshes have many topological differences. This means that the numerical results are not enough to reject H_0.

- Large areas of self-occlusion do not prevent reconstruction, but the results are more likely to be inaccurate. This method has no knowledge about the tree species' general shapes, and so does not make any educated guesses to fill gaps in the point cloud.
- The RANSAC algorithm can only avoid the influence of noise so much. When points sampled from leaves and twigs are packed tightly together with points sampled from the main branch, they usually cannot be avoided. Sometimes this results in a highly erroneous depiction of the target branch.
- Sketching in two dimensions will inevitably cause some inaccuracies, since the tested meshes are in three dimensions. Branches that are lined up will usually merge together into one branch in the reconstruction. Also, too many points behind or in front of a target branch may cause the center of the reconstruction to be off.
- The spherical sampling approach does not always simulate the results of real LIDAR scans. Often, the base of the trunk and the top of the crown were overrepresented in spherical sampling. Real LIDAR scans usually have fairly even coverage of samples.

6.3 Future Work

There are several avenues of future work that could be pursued to improve this method.

- Allowing reconstructions from multiple viewpoints to be registered together as a single mesh would help overcome the issue of working in two dimensions. For each viewpoint, the user could sketch out only the best represented branches.
- More granular control over the sampling size for each sketch node would help reduce the influence of noise. In addition, a visual representation of the sampling region on the user interface would help the user understand where noise might be originating from.
- A more accurate LIDAR simulator would help improve the results. In other words, the ray caster could follow the exact same patterns as the real-world laser scanner, and an additional function could be used to register these samples into a single point cloud.
- A study into the potential of this method as a compression tool could be performed. Because only the important vertices are recorded, the method could result in a dramatic decrease in file size.
- A user study could help determine this method's ease of use.

LIST OF REFERENCES

LIST OF REFERENCES

Aspert, N., Santa-Cruz, D., & Ebrahimi, T. (2002). Mesh: measuring errors between surfaces using the hausdorff distance. *Proceedings of the IEEE International Conference in Multimedia and Expo,* 1, 705 – 708.

Bucksch, A., & Lindenbergh, R. (2008). Campino – a skeletonization method for point cloud processing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1), 115 – 127.

Bucksch, A., Lindenbergh, R., & Menenti, M. (2009). Skeltre – fast skeletonization for imperfect point cloud data of botanic trees. *Eurographics Workshop on 3D Object Retrieval*, 13 – 27.

Chen, X., Neubert, B., Xu, Y.Q., Deussen, O., & Kang, S.B. (2008). Sketch-based tree modeling using markov random field. *ACM Transactions in Graphics,* 27(5). doi: 10.1145/1409060.1409062

Cheng, Z. L., Zhang, X. P., & Chen, B. Q. (2007). Simple reconstruction of tree branches from a single range image. *Journal of Computer Science and Technology,* 22(6): 846 – 858.

Cignoni, P., Corsini, M., & Ranzuglia, G. (2008). MeshLab – an open-source 3D mesh processing system. *ERCIM News,* 73, 47-48.

Cignoni, P., Rocchini, C., & Scopigno, R. (1998). Metro: measuring error on simplified surfaces. *Computer Graphics Forum,* 17(2), 167 – 174.

Côté, J. F., Widlowski, J.L., Fournier, R. A., & Verstraete, M. M. (2009). The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. *Remote Sensing of Environment*, 113(5), 1067 – 1081.

Cornea, N. D., Silver, D., Yuan, X., Balasubramanian, R. (2005). Computing hierarchical curve-skeletons of 3D objects. *The Visual Computer*, 21(11), 945 – 955. doi: 10.1007/s00371-005-0308-0

Decaudin, P. (2011). AntTweakBar (Version 1.14) [C++ Library]. Available from www.antisphere.com

Ferley, E., Cani-Gascuel, M. P. (1997). Skeletal reconstruction of branching shapes. *Computer Graphics Forum*, 16(5), 283 – 293. doi: 10.1111/1467-8659.00195

Fischler, M.A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 381 – 395. doi: 10.1145/358669.358692

Gorte, B., & Pfeifer, N. (2004). Structuring laser-scanned trees using 3D mathematical morphology. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35, 929 – 933.

Holmgren, J., & Persson, A. (2004). Identifying species of individual trees using airborne laser scanner. *Remote Sensing of Environment, 90*(4), 415 – 423. doi:10.1016/S0034-4257(03)00140-8

Huttenlocher, D. P., Klanderman, G. A., & Rucklidge, W. J. (1993). Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 15(9). 850 – 863.

Livny, Y., Yan, F., Olson, M., Chen, B., Zhang, H., & El-Sana, J. (2010). Automatic reconstruction of tree skeletal structures from point clouds. *ACM Transactions on Graphics, 29*(6). doi: 10.1145/1866158.1866177

Livny, Y., Pirk, S., Cheng, Z., Yan, F., Deussen, O., Cohen-Or, D., & Chen, B. (2011). Texture-lobes for tree modeling. *ACM Transactions on Graphics, 30*(4). doi: 10.1145/1964921.1964948

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., & Huebner, R. (2003). *Level of detail for 3D graphics.* San Francisco, CA: Morgan Kaufmann.

Morsdorf, F., Meier, E., Kotz, B., Itten, K. I., Dobbertin, M., & Allgower, B. (2004). Lidar-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management. *Remote Sensing of Environment, 92*(3), 353 – 362. doi:10.1016/j.rse.2004.05.013

Neubert, B., Franken, T., & Deussen, O. (2007). Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics, 26*(3).

Riccio, C. (2011). OpenGL Mathematics (Version 0.9) [C++ Library]. G-Truc Creation. Available at glm.g-truc.net

Pan, X., & Lane, D. M. (1999). Representation and recovery of 3D curved objects using generalized cylinders and the extended Gaussian image. *Pattern Recognition Letters*, *20*, 675 – 687.

Pfeifer, N., Gorte, B., & Winterhalder, D. (2004). Automatic reconstruction of single trees from terrestrial laser scanner data. *In Proceedings of 20th ISPRS Congress.* 114 – 119.

Prusinkiewicz, P., & Lindenmayer, A. (1990). *The algorithmic beauty of plants.* New York: Springer-Verlag.

Pyysalo, U., & Hyyppa, H. (2002). Reconstructing tree crowns from laser scanner data for feature extraction. *International Archives of Photogrammetry and Remote Sensing, 36*(1), 3 – 7.

Vanek, J. (2011). GluxEngine Framework (Version 0.9) [Computer software]. West Lafayette, IN: Purdue University High Performance Computer Graphics Laboratory.

Verroust, A., & Lazarus, F. (1999). Extracting skeletal curves from 3D scattered data. *Proceedings of IEEE Conference on Shape Modeling and Applications.* 194 – 201.

Xu, H., Gossett, N., & Chen, B. (2007). Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Transactions on Graphics, 26*(4). doi: 10.1145/1289603.1289610

Zhu, C., Zhang, X., Huand, B., & Jaeger, M. (2008). Reconstruction of tree crown shape from scanned data. *Technologies for E-Learning and Digital Entertainment,* 745 – 756. doi: 10.1007/978-3-540-69736-7_79

APPENDICES

Appendix A Data Structures

This section includes snippets of code from key data structures in the implementation. It is intended to provide supplementary material for chapter 4.

SketchNode class:

```
class SketchNode
{
private:
        // The node's location in 2D coordinates
        glm::vec2 coord;

        int nodeId;
        float lineEpsilon;
        slice * nodeDisc;

        list<SketchNode*> childNodes;

        // A pointer to the node's parent
        SketchNode * parentNode;

        // Data for point's projection into 3 space
        glm::vec3 projFront;
        glm::vec3 projBack;
        glm::vec3 * norm;

        // Data for RANSAC calculation
        vector<glm::vec3> nearPts;
        vector<Point<3>> testPts;
}
```

SketchTree class:

```
class SketchTree
{
private:
        ///head node of the sketch
        SketchNode * sk;
        ///id of active node
```

```
        int activeNodeId;
        ///sketch id count
        int nodeCount;

        // Collection of triangle vertices. Triangle vertices are grouped in threes.
        vector<GLfloat> * reconstructionVertices;

        // Reference to the point cloud
        vector<glm::vec3> * ptsRef;
    }
```

Slice struct:

```
    struct slice {
        glm::vec3 center;
        GLfloat radius;
        vector<glm::vec3> vertices;
    };
```

Appendix B Functions

This section includes snippets of code for key functions used in the implementation. It is supplementary material for chapter 4.

Model Intersection function:

```
glm::vec3 RayCaster::modelIntersect(const glm::vec3 & o, const glm::vec3 & d) {
	// o and d are the endpoints of the ray
	vector<glm::vec3> intersections;
	glm::vec3 pt;
	for (unsigned int i = 0; i < targetMeshCount; i++)
	{
		for (unsigned int j = 0; j < targetMeshes[i]->nfaces; j++)
		{
			pt = faceIntersect(o, d, targetMeshes[i]->faces[j], i);
			if (pt.x != 0 || pt.y != 0 || pt.z !=0)
			{
				// Make sure the zero vector was not returned
				intersections.push_back(pt);
			}
		}
	}
	// If the ray had multiple intersection points, find the one closest to
	// the ray's origin
	if (intersections.size() > 0) pt = nearestPoint(intersections, o);
	else pt.x = pt.y = pt.z = 0;
	return pt;
}
```

Collect Points function:

```
void SketchNode::CollectPoints(vector<glm::vec3> * ptsRef)
{
	nearPts.clear();
	for (unsigned int i = 0; i < (*ptsRef).size(); i++)
	{
		if (pointPlaneDistance((*ptsRef)[i]) < EPSILON)
		{
```

```
                                        nearPts.push_back((*ptsRef)[i]);
                        }
                }
                if (!childNodes.empty())
                {
                        SketchNode* temp;
                        for (childIter = childNodes.begin(); childIter != childNodes.end();
        childIter++)
                        {
                                temp = *childIter;
                                temp->CollectPoints(ptsRef);
                        }
                }
        }
```

Calculate Disc function:

```
        void SketchNode::CalcDisc()
        {
                testPts.clear();
                Point<3> p;

                do
                {
                        for (unsigned int i = 0; i < nearPts.size(); i++)
                        {
                                if (pointLineDistance(nearPts[i]) < lineEpsilon)
                                {
                                        p[0] = nearPts[i].x;
                                        p[1] = nearPts[i].y;
                                        p[2] = nearPts[i].z;
                                        testPts.push_back(p);
                                }
                        }

                        vector<double> ransacParameters;
                        Point<3> ransacCenter;
                        double desiredProbabilityForNoOutliers = 0.999;
                        CircleParamEstimator cEstimator(0.5);

                        double usedData =
                RANSAC<Point<3>, double>::compute(ransacParameters,
```

```
                                                        &cEstimator,
                                                        testPts,
                                                desiredProbabilityForNoOutliers);

                if(ransacParameters.empty())
                {
                        lineEpsilon++;
                }
                else
                {
                        projectDisc(ransacParameters);
                        return;
                }
        }
        while (testPts.size() < nearPts.size());
}
```

Appendix C User Sketches

This section presents the LIDAR point clouds and user sketches used to create the results seen in chapter 5.

Figure C.1 – Balsam Poplar user sketch.

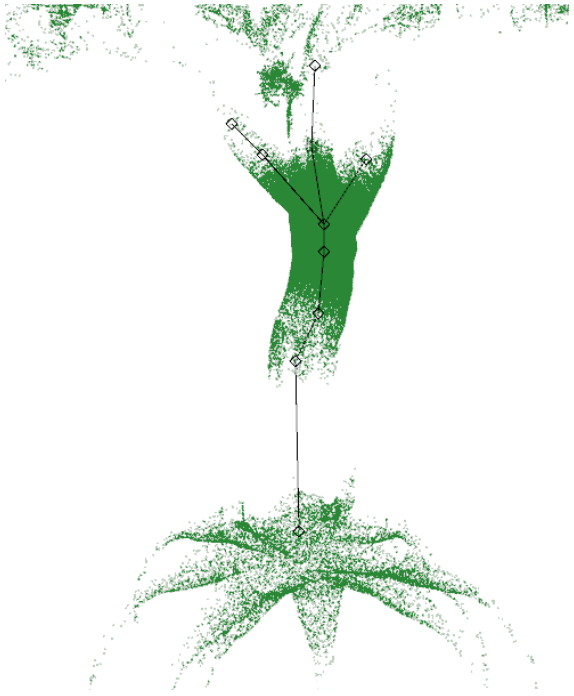

Figure C.2 – Bristlecone Pine user sketch.

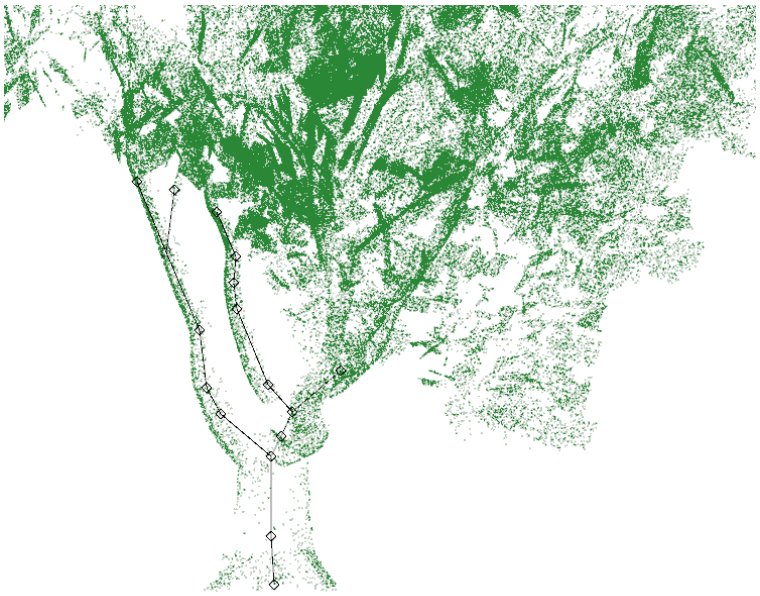

Figure C.3 – Coast Live Oak user sketch.

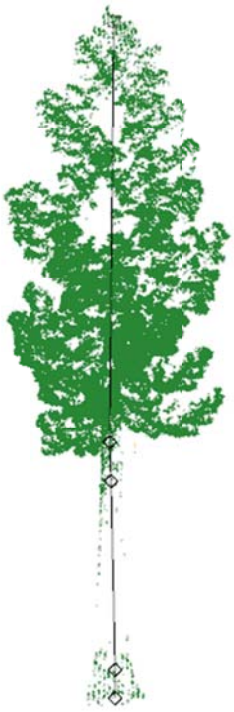

Figure C.4 – Giant Sequoia sketch.

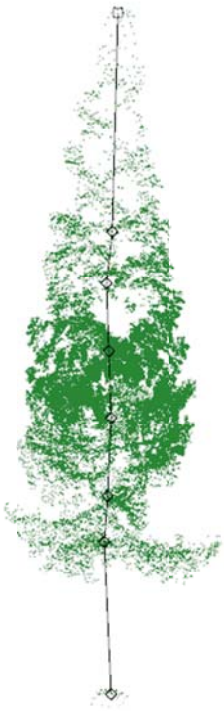

Figure C.5 – Grand Fir user sketch.

Figure C.6 – Little Walnut user sketch.

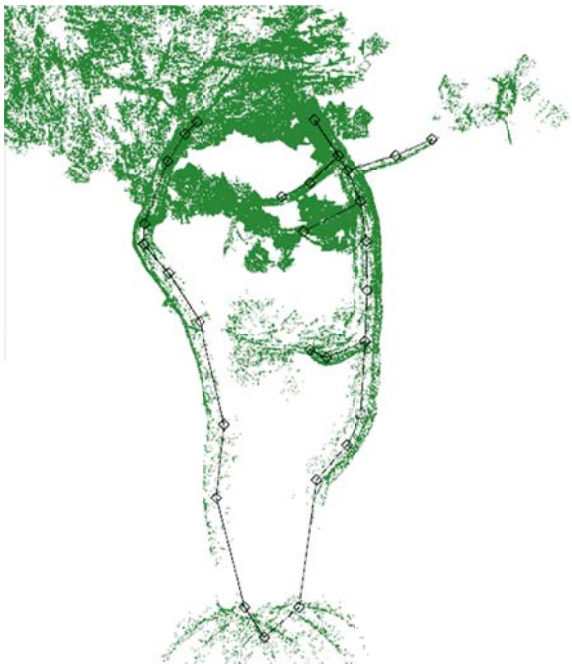

Figure C.7 – Monterey Cypress user sketch.

Figure C.8 – Oregon Ash user sketch.

Figure C.9 – Vine Maple user sketch